# Stateless: a Technical Report

Habla Computing

June 2017

**Abstract**

Abstract goes here!

# Contents

# Chapter 1

# Introduction

Software industry as we know it lacks well founded frameworks to program reactive systems[1] in a modular way. As a result, it is not hard to find monolithic software that entangles different computational aspects and which is quite difficult to reason about. Thankfully, *Functional Programming* (FP) provides such a wide range of modularity mechanisms, that it has recently become essential to deal with the increasingly complex demands made by society. *Domain Specific Languages* (DSLs) are key elements for that success. As their name suggests, these are languages that are exclusively focused on a particular domain, delegating the irrelevant low-level details to their particular interpretations. There are plenty of DSLs that conforms programs to describe a kind of system evolution, such as *SQL*, *Slick* or even *akka-http*. However, we have missed for a long time a DSL whose algebra was expressive enough to unify them all.

Coalgebra has been postulated as a unifying theory of systems. Indeed, automatas, streams and objects, among many other machines, could be represented by this abstraction. In spite of that fact, there seems to be little work on providing coalgebraic programming libraries *for the masses*. This might be a consequence of the blurred boundary between algebraic and coalgebraic worlds. In fact, any coalgebra could be seen as a state-based representation of an algebra, as we informally showed in our blog[2]. Starting from a lens represented as a (costate comonad) coalgebra, we ended up claiming that it could also be represented as a particular interpretation of *MonadState*. State and lens connections arise everywhere but this one led us to ask ourselves further questions. Mainly, if *MonadState* is the algebra for lens, what is the corresponding algebra for the rest of optics?

Once applied a similar derivation for each optic we collected a group of algebras and packed them in a Scala library, which we affectionately named *stateless*. Some of those algebras were actually widespread in the community, such as *MonadState* for lens or *MonadReader* for getter. However, some new algebras which were absolutely new to us emerged in the process. Most importantly, we noticed that our algebras could be casted and composed following the same rules that govern optic hierarchies[3]. However, composition was clumsy, manifesting an analogous situation to the one that arises when composing classic optics, eg. a lens represented with a pair of functions *get* and *set*. In that context, Van-Laarhoven and recently Profunctor representations offer a composable solution. Thereby, we tried to represent our algebras with a different approach, resulting in what we named *natural* representation, since it heavily relies on natural

[1] Reactive systems interact with their environment by receiving inputs and providing outputs, they run possibly forever and they show a black-box behaviour.

[2] https://blog.hablapps.com/2016/10/11/yo-dawg-we-put-an-algebra-in-your-coalgebra/ https://blog.hablapps.com/2016/11/10/lens-state-is-your-father/

[3] http://julien-truffaut.github.io/Monocle/optics.html

transformations. Indeed, natural transformations know how to turn a program over the focus into a program over the whole, where inner and outer programs are not necessarily the same. As a consequence, algebra composition becomes quite flexible and intuitive.

Optic algebras bring optic capabilities to a generic setting. The language they provide could be used to describe a system in a modular way, by establishing dependencies with nested subsystems in terms of dependencies with inner programs. Given that situation, we could specify that a *university* system is made of *department* subsystems, where traversal algebra is the optic that establishes the glue among both. Once the system is described, we could instantiate it with a naive state monad or with a mixing of Slick (for the university global information) and microservices (to keep every department updated) or with any other configuration we could think of. Undoubtedly, optics are key abstractions to achieve complex transformations of data. We claim that optic algebras achieve analogous complex transformations at a higher abstraction level, and therefore, they are suitable to unify different state-based DSLs. Besides, they are very close to an object-oriented mindset, since we can consider systems as classes where optics are the mechanism that determines dependencies and cardinalities with other classes. That is the reason why we suggest that stateless is a coalgebra-inspired library that could be potentially used by the masses to program reactive systems.

The rest of this paper is structured as follows...

# Chapter 2

# Background

## 2.1 Optics: Lenses and Beyond

## 2.2 Optic Representations

### 2.2.1 Classic

### 2.2.2 Van-Laarhoven

### 2.2.3 Profunctor

## 2.3 Optics and State Connections

### 2.3.1 BX Transformations

### 2.3.2 [Monocle] State Module

## 2.4 Initial Algebras (Free, Tagless)

# Chapter 3

# Body

## 3.1 Why Stateless?

## 3.2 Optic Algebras

## 3.3 Natural Representation

## 3.4 Operation Algebras (At, FilterIndex, etc.)

## 3.5 Standard Library (Map, etc.)

## 3.6 Indexed and Symmetric Algebras

## 3.7 Instances

### 3.7.1 Memory

### 3.7.2 Database

### 3.7.3 Microservices

# Chapter 4

# Example: University

# Chapter 5

# Conclusions and Future Work

## 5.1 Related Work

## 5.2 Discussion

## 5.3 Future Work